**AFRL-RI-RS-TM-2007-16**
**In-House Technical Memorandum**
**December 2007**


# SURVEY OF EVENT PROCESSING

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*

**STINFO COPY**

**AIR FORCE RESEARCH LABORATORY**
**INFORMATION DIRECTORATE**
**ROME RESEARCH SITE**
**ROME, NEW YORK**

# NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Air Force Research Laboratory Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TM-2007-16 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/ /s/

CHARLES G. MESSENGER
Chief, Situation Awareness Branch

JOSEPH CAMERA, Chief
Information & Intelligence Exploitation Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

# REPORT DOCUMENTATION PAGE

*Form Approved*
**OMB No. 0704-0188**



| 1. REPORT DATE *(DD-MM-YYYY)* | 2. REPORT TYPE | 3. DATES COVERED *(From - To)* |
|---|---|---|
| DEC 2007 | | Jun 07 – Aug 07 |

**4. TITLE AND SUBTITLE**

SURVEY OF EVENT PROCESSING

**5a. CONTRACT NUMBER**
In-House

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**
N/A

**6. AUTHOR(S)**

Thomas J. Owens

**5d. PROJECT NUMBER**
N/A

**5e. TASK NUMBER**
N/A

**5f. WORK UNIT NUMBER**
N/A

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
AFRL/RIED
525 Brooks Rd
Rome NY 13441-4505

**8. PERFORMING ORGANIZATION REPORT NUMBER**

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

AFRL/RIED
525 Brooks Rd
Rome NY 13441-4505

**10. SPONSOR/MONITOR'S ACRONYM(S)**

**11. SPONSORING/MONITORING AGENCY REPORT NUMBER**
AFRL-RI-RS-TM-2007-16

**12. DISTRIBUTION AVAILABILITY STATEMENT**
*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# WPAFB 07-0507*

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**
In the past decade, event processing technology has exploded from research at universities to a number of commercial products. In this paper, event processing technology will be reviewed, starting with the motivations behind its development and ending with a look into the future of event processing. A number of important topics closely related to event processing will also be examined, including applications of the technology, various academic and commercial implementations, event processing languages, and benchmarking techniques for event processing engines.

**15. SUBJECT TERMS**
Complex event processing, event processing languages, event process systems, data stream management, data mining

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | Nancy A. Roberts |
| U | U | U | UL | 23 | 19b. TELEPHONE NUMBER (*Include area code*) N/A |

# Table of Contents

# Introduction

Approximately 30 years ago, the relational database changed the information technology and information management communities. Today, another revolution is in progress in those same communities. This revolution is the design, implementation, and commercialization of event processing technology. Event processing will change the information technology and information management communities as much as, if not more than, the introduction of the relational database.

Most of the work involving event processing systems has occurred in the past decade. It has only been in the past five years, roughly, where event processing has transitioned from research to applications in a variety of industries. With growing numbers of companies offering event processing technology, along with the first open-source event processing engines, the event processing movement is not showing any signs of slowing down. It is highly likely that this technology is not a fad that will "die off" or "go away" soon, but rather a new component, or another piece, to data management that will affect the way organizations manage their data assets.

This paper is designed to allow anyone with an interest in event processing to understand the technology, its roots in previous technologies, the terminology used by the event processing community, and the current state-of-the-art in event processing technology and systems. Although designed for a more technical audience, this paper can be read and understood by many stakeholders of event processing.

The paper is organized as follows: The remainder of the Introduction section introduces the reader to key words, phrases, and concepts that relate to event processing, as well as to a brief overview of the history of event processing. The Applications section defines several application domains and use cases for event processing technology. Event Processing Systems examines the general architectures and requirements for an event processing system as well as exploring specific implementations, both academic and commercial, of event processing technology. Event Processing Languages is dedicated to exploring the various subscription languages designed for use in event processing systems. Additional Work lists related technologies and future directions for event processing technology, including some on-going research. Finally, the paper is concluded with a broad look at the topics covered and a review of the documents referenced in or related to this paper.

## *Terminology*

Throughout this paper, and other papers that discuss event processing, there will be a variety of terms used. These terms include "event," "event stream," "event cloud," "data stream management system," "complex event processing," and more. At this time, these terms are not standardized. However, there is an ongoing effort, led by Dr. David Luckham and his working group that intends to standardize the terminology used by the event processing community [1].

For the purposes of this paper, definitions for key words, phrases, and concepts are provided. Many of these definitions come from the working group's efforts, while others are adopted from other documents relating to event processing. The purpose of this paper is not to (re)define words, but instead clarify any confusion that the reader might have.

The first notion is that of an event. The dictionary definition of "event" is "something that happens or is regarded as happening." In event processing, there are two definitions of event. The first relates to real-world occurrences, while the second involves representations of

those occurrences. Simply, an event is something that happens. At the same time, that occurrence is represented in some form for processing, and this representation is also called an event.

Often, two types of events will be mentioned. These are "simple events" and "complex events" (sometimes called "composite events"). A simple event is a single event that does not represent other events and is not composed of other events. A complex event is an abstraction or aggregation of events, either simple events or other complex events. There are two examples that can be used to help clarify exactly what simple and complex events are. The first example comes from the financial world, specifically the stock market. This is an example that is often presented in papers discussing event processing. The price of an individual stock at a specific time in the day is a simple event. However, the series of prices throughout the day (or perhaps even a longer period of time) for one or a series of stocks can be used to generate a complex event that represents various trends in the stock markets, and these complex events can be used to provide alerts to investors. This shows a complex event that is an aggregation of other, simple events. Another, more technical, example is that of a network monitoring system. In this example, events are system status reports from various networked machines. A simple event might be a printer reporting that it is out of paper. However, not all systems might have a need to know that a printer is out of paper. Rather, a complex event could be generated that represents a printer failure or even just a hardware failure, along with the location of the hardware that has failed. This is an example of a complex event that is an abstraction of a simple event.

Events are often said to exist in "event clouds" or "event streams." An event cloud is a partially ordered set of events that can be bounded or unbounded. An event stream is a linearly ordered sequence of events. Usually, the ordering is temporal, but any relationship can be used to provide ordering to events. An event cloud can be made of up event streams, along with any other event sources. One could view an event cloud as all of the data sources accessible by an operation, both internal to the operation and external, including data feeds, data streams, data files (flat files or XML files), databases, and more. This set of data sources could include data that is important to ongoing operations, as well as noise, or data that is not relevant to the situation. Each entry in a feed, stream, or file could be considered an event, especially if it has a timestamp indicating when it occurred.

There are a variety of names for the processing of events. These names include "complex event processing," "event stream processing," "data stream management," and "streaming data management," among others. In application, these names describe a variety of systems that are designed to process events. The differences are the event source, the scalability, and the types of event data that can be processed. For the purposes of this paper, the terms "event processing engine" (sometimes "engine") and "event processing system" (sometimes "system") will refer to these types of systems.

One component of event processing is the ability to perform operations on events. This process is often described as either "continuous querying" (sometimes just "querying") or "subscription." The word "subscription" is a much better description of the actions taking place. Query brings up the thought of requesting data. However, in event processing, data is not requested, but rather, it is generated and provided, especially when dealing with streaming data. The term subscription conjures up images of having information delivered, or even having to go extract the data from a location when the source wants you to have it. This is more in line with the event processing field, where information is pushed from the source or sometimes stored in a source and must be pulled by any interested parties before it is removed from the source.

Throughout this document, the word “subscription” and “subscription language” will be used where some will often use “(continuous) query” or “query language.”

### *A Brief History of Event Processing*

Within the past few years, the event processing community has grown. The use of weblogs and interest groups has led to this growth. Today, there are a number of blogs and groups [2] [3] [4] dedicated to exploring event processing. Although some of these groups are led or sponsored by vendors, others are vendor neutral (although vendors do communicate on these channels). These communities are excellent sources of information, especially on the history of event processing. The Complex Event Processing Resource Site [5] offers information about event processing, including a brief history of the field [6].

Event processing is rooted in academic research, which started in the early to mid 1990s. The leading researchers were Dr. K. Mani Chandy at Cal Tech, Dr. John Bates at Cambridge University, and Dr. David Luckham at Stanford University. The focus of this early research was on the processing of streaming event data, and more specifically to identify complex sequences of events and to control actions that occur in response or because of these sequences, or patterns.

By 2000, the first two commercially-available event processing technologies were released. Dr. Chandy’s work led to iSpheres, which would later be acquired by Avaya. Dr. Bates, together with Giles Nelson, started Apama. At about the same time, academic research was ongoing in the field. Stanford University established the STREAM project [1], while Massachusetts Institute of Technology, Brown University, and Brandeis University began to collaborate on the Aurora [8], and later the Borealis projects [9].

Shortly after these events, in 2002, Dr. Luckham would publish the first book on event processing [10]. Within a year of the publication of Dr. Luckham’s book, two more event processing technologies would become commercially available. These systems, which are still available, are Coral8 [11] and StreamBase [12]. Both have their roots in the STREAM and Aurora/Borealis projects. Specifically, Dr. Michael Stonebraker, who was heavily involved in the Aurora and Borealis projects, founded StreamBase systems while Dr. Rajeev Motwani, the co-leader of the STREAM project, is the chief scientist and co-founder of Coral8, Inc.

Other projects at universities were also in-progress. The most notable of these projects was TelegraphCQ [13] at the University of California at Berkeley. This system would be the basis of the Truviso commercial product as Dr. Michael Franklin and Dr. Sailesh Krishnamurthy, as a graduate student, were also involved with the TelegraphCQ project at UC Berkeley.

Today, there are a number of academic and commercial projects. The most notable commercial projects include work by AptSoft [14], BEA Systems [15], and EsperTech [16]. Other notable academic projects, both ongoing and concluded, include StatStream [17], COUGAR [18], and Cayuga [19]. Some of these systems will be discussed in more detail in later sections, while others are beyond the scope of this paper.

## Applications

There are a number of applications where data is streamed. In most, if not all of these areas, event processing technology can be used to improve operations. Some of these domains include sensor data, internet traffic, financial tickers, online auctions, web usage logs, and telephone call records [20], just as an example of the varied application domains.

A number of interesting applications for event processing technology are discussed in this section. However, this is by no means a complete summary of how event processing can be

deployed. Deployment is largely determined by domain specialists working alongside computer scientists, software engineers, and information technologists. Only cooperation between software specialists and domain specialists will allow the full extent of event processing to be realized.

## *Data Analysis and Mining*

There are many specific applications for data mining. One area of particular interest to both the data mining and event processing communities is the application of event processing systems to the mining of system data logs, especially in real-time, using streaming system log data [21]. However, it is believed that any data mining operation can be improved using the research and information from the system data log mining experiments.

System data log files can be extremely complex and contain large amounts of data. Some systems do not even fully utilize logging systems and even more finely-grained data can be produced. However, even the current amount of data generated is far too much for a human operator to analyze, so automated techniques are being developed. One goal of the system log analysis community is to be able to predict system errors and failures before they happen, using the data generated by system logging mechanisms. This would save organizations time, money, and resources.

The first step in real-time data analysis is to be able to preprocess the data generated by a variety of systems into a format that can be easily processed. The preprocessing that is required includes sampling, cleaning, the addition of new attributes, fusing streams from a variety of sources, and supporting the capability to run multiple algorithms over the data obtained. This is required before any mining or analysis algorithms can be applied. A scalable, modular, distributed architecture is the focus of current investigations. A current approach is to combine statistical learning theory algorithms and an event processing system.

An early prototype [21] started with a decision tree algorithm and was expanded. The goal was to take data streams and files as input and produce streams as output, and allow the output streams to contain information that could be of interest to various clients. This system was also designed to support a variety of analysis algorithms using a “wrapper” model for reading the streams. The future goals are to investigate new algorithms that could be used for mining and analysis as well as to test the algorithms over both simple streams, such as those from sensors, and more complex streams, such as those from system logs.

Another concern for the data analysis and event processing communities is the identification of unknown event patterns [22]. Many systems support the ability to detect specified patterns, using either pattern templates or the subscription language. However, the ability to detect new and different patterns in a series of events could be useful in some operations. Heuristics can not be applied in this situation; however event processing agents and algorithms can be applied.

There are a number of algorithms that could be useful. Some of these algorithms include deterministic or probabilistic approaches, discriminant analysis, Bayesian belief networks, hidden Markov models, and more [22].

## *Intrusion Detection Systems*

Intrusion detection systems are of great interest to network system administrators [23]. Intrusion detection, simply, is the ability to detect an attempt to compromise the confidentiality,

integrity, or availability of a system. This is especially important in systems critical to operational success.

Event processing technology allows for an improvement to existing intrusion detection systems, as well as leading into improvements in intrusion prevention. Information about a network system comes from a variety of sources, including system log files, existing intrusion detection systems in place on a client and/or host level, network traffic monitoring systems, and network statistics. Each of these data sources provides information that can be used to detect and possibly prevent intrusions into network systems.

The first priority is to fuse data from available sources to produce a data stream that contains real-time data about the network. Once fused, the resulting data stream can be monitored against known patterns that occur when the system is being compromised. Upon the detection of such a pattern, there are multiple actions that can be taken. One possible course of action is that the system administrators are notified of a possible intrusion and provided with the data that is relevant to attempt to stop the attempts. Given the nature of the system, it might also be possible to issue automated commands to protect systems from those that might be compromised.

## *Sensor Management*

Sensors are becoming more common in today's world. These sensors are consistently producing data about their physical surroundings. In a sensor network, there could be any number of sensors, each producing a measurement of a different nature in a different format. The management of not only the data, but also the type of data produced by each sensor is important.

There are two approaches to managing such sensor networks [24]. The first is to create a centralized architecture. Data collected from every sensor is collected and stored in a central repository, usually using a traditional database management system. An alternative option is to use a data stream management system to manage the data produced by sensors in real-time and allow for queries over the data produced by the sensors that can be evaluated as data arrives.

## *Situational Awareness*

Event processing technologies have been investigated for application in systems that allow for situational awareness [25]. Awareness means knowing the events that are unfolding in a situation so that actions can be taken in response to those events. There are different kinds of relevancy – contextual, situational, and temporal. Information that is contextually relevant relates to concepts familiar to the user, situational relevancy means that the information is relevant to the tasks assigned to the user and the role of the user in the organization, and temporal relevancy means that any data or information is given to the user on-time.

Using events to provide situational awareness involves being able to describe real-world occurrences using events as well as understanding the relationships between these events. Event pattern matching and the detection of unknown patterns would be beneficial to implementing an event processing engine in a situational awareness setting. Once the situation is understood and events are detected as they occur, actions can be taken, either by a human or by a system, to take advantage of the situation or to minimize the risks associated with the situation.

# Event Processing Systems

There are a large number of event processing systems, both academic and commercial. It is beyond the scope of this paper to discuss every system in detail; however it is important to understand the systems that are of note to the event processing community. There are some academic and commercial systems of interest, either for historical reasons or for introducing a new concept to the event processing community.

There are two models for data and information management – human-active, database management system passive (HADP) and human-passive, DBMS-active (HPDA) [26]. In the HADP model, a database is a passive repository for large amounts of data. Clients, which are often humans using a computer system to interact with the database, initiate queries and transactions. These systems are not designed for the real-time processing capabilities of an event processing engine. The HPDA model, although "DBMS" does not adequately describe the properties of an event processing engine, can be used to implement an event processing engine. This model allows for the data processing engine to analyze the data as it is provided and alert a client, which could be a system or a human operator, after certain conditions have been met.

## *System Requirements*

There are certain requirements that must be met by a system in order for it to be considered an event processing engine. These requirements include high availability, scalability, and optimizations. Dr. David Luckham developed a "checklist" of capabilities for event processing tools [27]. The capabilities include real-time computation on the data contained within an event, the ability to detect and possibly react to simple event patterns, process streaming event data, detect and possibly react to complex event patterns, and the ability to abstract event patterns. Not all systems have to have all of these capabilities, but must have at least one of these.

High availability means that the system is available for processing and has a very high uptime. There have been efforts to determine the most efficient and effective methods for maintaining the high availability of data management systems, and specifically event processing systems [28]. The critical component of high availability is having a backup that can take over processing functions quickly and without data loss.

Optimizations are also very important in event processing systems [20]. System resources are limited and clients have a desire for answers in near-real-time or even in real-time. Because of this, optimizing a system is essential, at both the resource and the processing levels. Optimization also connects to the availability of a system and results to the system. In terms of data loss, systems have different tolerances. Some systems are capable of handling missing and/or incomplete data, while it is essential that some systems have all or nearly all of the data. It is important for system designers to know how tolerant clients are to data loss. Quality-of-Service (QoS) must be maintained, even if some data must be dropped to maintain the real-time requirements or reduce demand on the available resources.

A final requirement to account for is scalability. There are a potentially large number of people requesting information from a large number of data sources, each transmitting a potentially large number of events in a small timeframe. There are multiple approaches to dealing with scalability, most relating to the architecture of the event transmitters and event processors.

When attempting to satisfy the requirements, tradeoffs must be taken into account [29]. There are three factors that affect a system – event safety, scalability, and expressiveness. Event

safety refers to exchanging information about an event without revealing internal representations of that event. There are three main conflicts, which are event safety versus scalability, expressiveness versus scalability, and event safety versus expressiveness. As data representation becomes more powerful and safer, the languages used to register subscriptions/queries should become more expressive. However, highly expressive languages can limit the scalability of the system. Highly expressive languages allow any interested party to provide specific interests, but this increases the processing time required for a system. Finally, if the language allows for custom event typing, it becomes even more difficult to ensure type safety.

These are the major considerations during the implementation of an event processing engine, but these are not all of the considerations. In the following sections, specific implementations, academic and commercial, will be examined.

## *Academic Systems*

### Aurora and Borealis

The Aurora and Borealis systems are closely related systems. Both are described as a general-purpose data stream management system [30] in the papers published by the creators at Brandeis University, Brown University, and the Massachusetts Institute of Technology. The goal of the systems is to support various real-time monitoring applications.

The overall system architecture of Aurora and Borealis is based on the "boxes-and-arrows" process- and work-flow systems [31]. Data flows through the system as tuples, along pathways, which are arrows in the model. The data is processed at operators, or the boxes. After the last processing component, they are delivered to an application for processing.

One of the key components of Aurora and Borealis is the notion of quality-of-service (QoS). Functions or graphs are used to define results in terms of performance or quality [30]. There are three types of graphs used in the systems – latency graphs, value-based graphs, and loss-tolerance graphs. The purpose of quality-of-service is to allow for performance optimizations that continue to allow output that is within tolerance of the applications receiving the data.

There are several optimizations that these systems are capable of carrying out to decrease system stress. The primary optimizations are the insertion of processing boxes, moving processing boxes, combining two boxes into a single, larger box, reordering boxes, and load shedding [30]. Load shedding is one of the most important optimizations introduced in any event processing systems. Load shedding means that the number of tuples presented for processing are reduced to end overload states. With Aurora and Borealis's notion of quality-of-service, load shedding is done in a manner that minimizes losses and impact, opting to drop the tuples relating to systems that are more tolerant of lost and missing data.

Aurora was the first system developed at Brandeis, Brown, and MIT. Borealis is the second generation system developed by these three universities [32]. The stream processing functionality of Borealis came from the Aurora system, while the distribution techniques came from a project known as Medusa [33].

The source code for Aurora is available at the Aurora Project Homepage [8], and the Borealis system is available upon request through the Borealis Project Homepage [9]. It should also be noted that some of the Aurora team has commercialized the Aurora project through StreamBase [12] [31].

## Cayuga

Cayuga is a newer event processing system in development at Cornell University. This project is part of 2007 AFRL/IF/AFOSR Minigrant titled "User-Centric Personalized Extensibility for Data-Driven Web Applications," by James Nagy (AFRL/IFED) [34]. This minigrant focuses on Cayuga as a stateful publish/subscribe system for use in a graphical programming model (also being developed at Cornell) known as Hilda. An overview of both systems can be found in the Minigrant Proposal.

Researchers at Cornell describe Cayuga as a general-purpose complex event processing system [35]. The system can be used to detect event patterns in event streams. The Cayuga system is designed to leverage traditional publication/subscription techniques to allow for high scalability [36]. This leads to comparisons not only with other data stream management systems, but also to publish/subscribe systems to demonstrate the applications and capabilities of Cayuga.

One of the most novel components of Cayuga is the implementation of the processing engine, which utilizes a variation of nondeterministic finite automata [35]. However, the automata in Cayuga are a generalization on the standard nondeterministic finite automata model. These automata read relational streams, instead of a finite input alphabet. Also, the state transitions are performed using predicates. The use of automata allows for the storing of input data and new inputs can be compared against previously encountered events.

At this time, the Cayuga source code is not available to the public. Distributions received at the AFRL are, according to Cornell researchers, pre-alpha and alpha releases. Work is expected to continue on the project to produce a beta version by the end of 2007. Additional information regarding Cayuga can be found on the Cornell Database Group's Cayuga webpage [19]. There is also research regarding a distributed implementation of Cayuga, known as FingerLakes [37].

## STREAM

STREAM is the Stanford Data Stream Management System, produced at Stanford University [37]. The goal of STREAM is to be able to consider both structured data streams and stored data together. The queries over data streams are issued declaratively, but are translated into flexible physical query plans. Other approaches taken in the STREAM system include adaptive approaches to processing, load shedding, and the ability to provide approximate answers, and manipulations of query plans during execution.

One of the notable features of the STREAM system is its subscription language, known as the Continuous Query Language, or CQL. CQL features two layers – an abstract semantics layer and an implementation of the abstract semantics. The abstract semantics defines three operators and two data types, each with specific properties. The implementation uses SQL to express relational operations and adds extensions for stream-related operations.

According to the STREAM homepage, the project has officially wound down. The source code is available for download through the team's website [7]. Some key members of the STREAM team have moved to Coral8, Inc. where they are now working on the Coral8 event processing engine.

## TelegraphCQ

TelegraphCQ, from the University of California at Berkeley, is designed to provide event processing capabilities alongside relational database management capabilities by utilizing the PostgreSQL open-source code base [40]. The existing architecture of PostgreSQL is modified to

allow for continuous queries over streaming data. Several components of the PostgreSQL engine underwent very little modification, while others were significantly changed. The most significant component of the TelegraphCQ system is the "wrapper," which allows for data to be pushed or pulled into the Telegraph processing engine, and custom wrappers allow for data to be obtained from any data source [41].

The TelegraphCQ source code for versions 0.2, 2.0, and 2.1 are available for download from the TelegraphCQ website [13]. The system has also been commercialized as the Truviso event processing system.

## *Commercial Systems*

### BEA WebLogic

BEA Systems' entry into the event processing field occurred in mid-2007 with the introduction of their WebLogic Real Time and WebLogic Event Server systems. The Air Force Research Laboratory Information Directorate uses BEA WebLogic technology in several ongoing projects and has a site license for WebLogic. On 07 August 2007, BEA Systems provided a presentation to the AFRL regarding updates to their WebLogic software suite, including the new Real Time and Event Server systems.

BEA Systems focuses on enterprise-level system architectures and service integrations. More specifically, their Event Server technology is a focus on event-driven service-oriented architecture which provides a response to events in real-time. As part of the package, they provide a complete event processing and event-driven service-oriented architecture infrastructure that supports high-volume, real-time, complex, event-driven applications. This is one of the few commercial offerings of a complete, integrated solution for event processing and service-oriented architectures.

For development, the WebLogic system includes a series of Eclipse-based developer tools as well as the ability to integrate dashboards and monitoring solutions. Administration tools for monitoring throughput, latency, and other statistics is also provided. Non-programmatic interfaces are also provided to allow all interested parties to configure queries and rules used for processing event data.

The BEA website [15] provides documentation and overviews of their technology. There are also several ongoing projects at the AFRL using WebLogic components. Benchmarks are also available.

### Coral8

The Coral8 event processing tool is designed to process multiple data streams, heterogeneous data streams, streams with high incoming data rates, processing operations that require filtering, aggregation, correlation (including correlation across streams), pattern matching, and other complex operations in near-real-time [42] [43]. In order to accomplish this, the Coral8 Engine is composed of two tools, which are the Coral8 Server and the Coral8 Studio, as well as software development tools, such as a Software Development Kit (SDK).

The Coral8 Server is the processing heart of Coral8 [42]. This highly optimized server can be executed on Windows, Linux, and UNIX operating systems. Clustering is also supported for greater performance of the engine [44]. Additional features of the server include the publication of a status data stream that can be used to monitor performance and activity of the

server as well as allowing for Simple Network Monitoring Protocol (SNMP) to be used by management consoles and frameworks for the monitoring of Coral8.

Coral8 Studio [44] allows for queries and data streams, both input and output, to be added and removed. Administrators can also use the Studio to monitor both Coral8 Servers and application components. The subscription language used in the Studio is the Continuous Computational Language, or CCL. The Studio provides an IDE-like interface to allow for queries, and data streams, to be added.

The Coral8 SDK supports C/C++, Java, .NET, Perl, Python, SOAP, JMS, and more [44]. This allows for the Coral8 Server functionality to be accessed programmatically, using any languages supported by the SDK. In addition, new data types can be supported by creating custom input and output adapters.

## Esper and NEsper

Esper and NEsper are well discussed on the EsperTech website [16]. Both Esper and NEsper are free and open-source (FOSS) software produced by EsperTech. Esper is a Java implementation and NEsper is a .NET implementation, written in C#. Both contain similar features and are maintained in parallel.

The performance reports on Esper [45] show very good performance. A number of tests were performed using a stock ticker simulation with 1000 symbols. Various statements were executed to obtain statistics. The results shows that latency was usually well below 10 microseconds. CPU usage was also relatively low, staying below 80%. However, the technical specifications of the system should be noted. The clients were run on machines using two Intel Xeon 5120 2GHz processors for a total of four cores and contained a total of four gigabytes of RAM. The JVM used to execute Esper was not the Sun JVM, but the BEA JRockit R27.3 JVM.

Unlike most other implementations, this software is not rooted in any particular academic research effort. Instead, the majority of the work is in-house. EsperTech allows for their software to be dual-licensed for use in commercial products. An example of this is the use of Esper in BEA Systems' event processing and real-time technologies.

## Progress Apama

According to the company's history [46], the Apama architecture dates back to the mid-to-late 1990s. The Progress Apama event stream processing platform [47] consists of several tools, including an event processing engine, data stream management tools, event visualization, adapters for converting external events into internal events, and development tools.

The Apama technology was tested at the Air Force Research Laboratory by Robert Farrell (AFRL/IFSA). These tests were designed to prove or disprove the marketing claims of Apama, relating to the throughput and latency. The results showed that Apama could process events at rates measured in thousands of events per second.

## StreamBase

The StreamBase event processing engine is based on research from the Massachusetts Institute of Technology, Brown University, and Brandeis University, or the Aurora project [48]. The goal of the product is to do the same for real-time data what relational databases and SQL presently do for stored data.

StreamBase provides a Server and a Studio module [49]. The Server module is designed to be able to scale from a single-CPU to a distributed system, including support for 64-bit

processors. The Studio is Eclipse-based and provides both graphical (drag-and-drop) creation of queries, but also supports text-based editing of the queries, which are issued using StreamSQL. Connectivity to outside data sources is permitted through ODBC and JDBC drivers, as well as Java and C/C++ interfaces and adapters to messaging systems.

### Truviso

Truviso is a commercial event processing engine [50] that is based on the research toward the TelegraphCQ project at UC Berkeley. The "claim to fame" for Truviso is that it supports a fully-functional SQL, and integrates PostgreSQL relational database alongside a stream processing engine. The integration of PostgreSQL leads to other aspects of the Truviso system.

The queries are simply standard SQL with extensions that add functionality for time windows and event processing. Carried over from PostgreSQL are user-defined functions, as well as JDBC and ODBC interfaces. In addition, the use of an integrated relational database allows for easy caching, persistence, and archival of data streams, as well as queries that include not only real-time data, but also the historical data.

### *Other Systems*

These are not the only systems that have been produced. There are continuing efforts, both academic and commercial to research and test new concepts in event processing. Some systems of note include COUGAR from Cornell, StatStream from New York University, and a growing number of commercial systems.

Many of these systems, especially the academic implementations, are focusing on different areas in event processing. For example, StatStream [17] [51] is designed to compute statistics in constant time, and use cases for financial applications have been demonstrated. COUGAR [18] [52] [53] [54] is designed as a solution that focuses on constraints in sensor networks to provide a method and architecture for managing sensor networks.

In the future, as event processing becomes more widespread, it is likely that even more applications will be developed to test new strategies, new ideas, and new problem domains. This will be in addition to general event processing, a category that most of the previous systems fall into. Work is far from completed on system design and implementation.

## Event Processing Languages

There have been a number of efforts toward various languages for use in event processing systems to allow users to express their interests. Different projects have taken different directions with the research. One approach is to modify a relational language to treat data streams as "append-only" relations and produce results in the form of tuples as soon as they become available [55]. Other directions include creating a new language with a familiar syntax or starting with real-time processing languages and extending or modifying them.

Event processing languages generally have some common characteristics [56]. These characteristics include the ability to extract data associated with an event, provide support for complex events, allow for temporal conditional statements, and detect event patterns. Not all languages have all characteristics, and some languages are stronger in certain areas than others. However, there are some behaviors that are expected when using an event processing system that are provided through the system's language.

These approaches will be discussed in terms of various implementations of event processing languages. Currently, there is no accepted method for implementing a subscription

language. There is still work to be done to determine how to design and then implement a language that is acceptable to the parties involved, both system designers and end users.

## Cayuga Event Language

The Cayuga Event Language [35] [36], abbreviated as CEL, is based on an event language instead of a more powerful query language. It is also a mapping of the operators in the algebra developed for the Cayuga system into a language that has a syntax modeled after SQL. This event language is tied in directly to the processing model of Cayuga.

The event language used to implement CEL is much weaker than relational languages; however it was designed for real-time data. It contained constructs that allowed for filtering and sequencing. Extensions for other functionality, such as parameterization and aggregation, were added to the event language. The semantics were well-defined using the Cayuga algebra and the implementation was highly optimized.

Left-associated expressions are implemented within the Cayuga system by the system's variation on nondeterministic finite automata. If the expression is not left-associated, then it is broken down into a series of left-associated expressions and implemented by a series of automata.

Another advantage of the CEL is the representation of the queries. Queries can be represented in CEL, which has a SQL-like syntax, or in an XML-based format called the Automaton Intermediate Representation, or AIR. AIR not only stores the queries, but also representations of the automata and edges, including their states.

## Continuous Query Language

The Continuous Query Language, or CQL, is supported by the STREAM system, produced at Stanford University [57]. This language also influenced the Continuous Computational Language, or CCL, which is the proprietary language used in the Coral8 event processing engine.

The CQL is based in SQL and is built on three public operator types as well as system operators. The public operators are stream-to-stream, relation-to-stream, and relation-to-relation. The relation-to-relation operators utilize the SQL functionality, while the stream operators are extensions and modifications.

The core feature of CQL is not the implementation, but the well-defined semantics used by the implementation. These semantics can be implemented by any language in any syntax, as long as the operator and data type rules are applied. This allows for a thorough understanding of the queries that can be issued using this language.

## StreamSQL

StreamSQL [58] [59] is an extension to SQL that is supported by StreamBase Systems for their StreamBase product. There is extensive documentation as well as a list of frequently asked questions available on the StreamSQL site.

Simply, StreamSQL is SQL that has been enhanced to process streaming data. All of the original functionality of the original SQL is maintained, but new querying capabilities are added to allow for the processing of streaming data. The basis for the StreamSQL language is in the STREAM project at Stanford as well as the Aurora project the Brandeis University, Brown University, and MIT.

There have been attempts to make StreamSQL the event processing standard. However, the general movement in the event processing community is away from SQL and toward languages designed for processing real-time data and event streams.

# Benchmarking Event Processing Systems

## *Linear Road*

The Linear Road benchmark [60] [61] system is designed to measure the performance characteristics of not only event processing systems, but also alternative systems, such as relational database systems designed to simulate event processing systems. This benchmark is currently endorsed by the development teams of the Aurora and STREAM systems.

Linear Road simulates a variable tolling system on expressways in a metropolitan area. Features such as accident detection and alerts, measurements of the traffic congestion, toll calculations, and historical queries from simulated drivers on the expressways are incorporated into the simulation. Certain criteria are also specified, such as the ability to maintain historical data and produce calculations on live data.

All required components of the benchmark, including the data generator, driver, and validation tools are available for public download from the Linear Road project website. Documentation is also available from the same source.

## *NEXMark*

The Niagara Extension to XMark [62] [63], or NEXMark, is an extension to the XMark benchmark for XML repositories. This benchmark is still in a draft phase, however it is well documented and explained in both the paper and on the NEXMark website.

NEXMark extends the XMark benchmark to an online auction system. At any given moment during the simulation, there are hundreds of open auctions. New users are registering with the system, new items are being auctioned, and bids arrive on the items being auctioned. NEXMark evaluates how well an event processing system handles this data flow, focusing on how fast the data is processed and how accurate the results are.

The two metrics used are the output matching metric and the tuple latency metric. The output matching metric measures how quickly and how accurate the results produced by the system are. Various answers can be considered to be correct, depending on the system and its processing, especially when dealing with data overload and load shedding. This measures how close to accurate the answer is as well as considering how long it took the system to produce that answer. The tuple latency measurement is less important, in many cases, as it simply measures how long it takes for a tuple to go from the system entrance to leaving the system.

## *BiCEP*

The BiCEP project is out of the University of Coimbra (Coimbra, Portugal) [64] that began in June 2007. The goal of the project is to attempt to overcome the lack of standards in the event processing community, identify the core requirements of event processing engines, and develop a benchmark that allows for products and algorithms to be compared even with their architectural and semantic differences. The creators of the BiCEP project are working under the assumption that the benchmark will actually evolve into a set of domain-specific benchmarks, each with their own data sets and query specifications

A number of metrics have been identified that BiCEP will attempt to measure. These include the throughput, response time, scalability, and adaptivity, among others. The impact of out-of-order events, event record size, past history comparisons, and query priority will also be examined.

Unlike Linear Road and NEXMark, BiCEP is vendor independent. There is no academic research or corporate backing to this project by any producer of event processing technology. This allows for a highly objective and independent assessment of event processing systems.

## Data Sets

The two main data sources for the event processing community are real world data sources and the stream query repository [39]. There have been some considerations on the various blogs and interest groups, especially in the CEP-Interest group [4] regarding generation of event clouds and data sets for use in testing event processing systems.

The primary uses for data sets are benchmarking and presenting contextually valid simulations to parties interested in event processing technology. However, there are many questions as to how to produce data sets that can, or should, be accessed by the public and how to implement the simulations using these data sets.

The first problem is to find or generate data sets. In order to make the results accessible by all, it is important to make sure that the data found in the sets is understood. An example would be the examples rooted in the financial world in various papers. Not everyone, technical and non-technical, understands the context of this data. Therefore, demonstrations using this data might have little to no meaning to people in other fields, such as the intelligence community or network administrators. These groups want to see information that they can relate to and how event processing technology can help them analyze it.

The next problem is choosing what data to include in the data set. A set could just include the event data and be processed as fast as the system can. This is not always the best solution, as it does not show the event processing engine under normal operating conditions for a particular use case. Instead, it might be necessary for data sets to be time-stamped and processed in real-time or a faster simulation of real-time (such as every real second is one minute or one hour in the simulation). This would provide much more accurate results and benchmarks that the end users can understand.

The final, serious problem is how to generate events that make sense in context, especially if there are no available data sets from the problem domain. Events must make sense; for example, if the movement of people is being simulated, then the simulated people should not move in a manner in which people would not (be able to) move. At the same time, there might be simulated noise in the events, where the noise is data that could be “bad” or somehow corrupted, especially in an environment where data might not always be clean.

So far, the problem of generating data sets has not yet been adequately addressed. Some of the work related to benchmarking event processing systems is also working on the generation of data sets for the purposes of generating the benchmarks.

# Conclusions

## *Related Work*

After looking at event processing, it is also important to understand technologies that are related to it. Work by Microsoft Research [65] discusses a vision for unifying three concepts – event processing, data streams and event streams, and asynchronous messaging. There are also efforts to better understand and utilize various components of event processing, such as event representations and system architectures.

The primary work that is related to event processing is the publish/subscribe paradigm [66]. The word event is often used when discussing publish/subscribe systems to refer to the information presented. There are a number of variations to the publish/subscribe system including topic-based, content-based, type-based, and semantic-based [67]. The differences between these variations are how events of interest are specified.

There are also various communication paradigms that are related to the publish/subscribe paradigm [66]. These include message passing, remote procedure call, notifications, shared spaces, and message queuing. These paradigms are sometimes considered during the design and implementation of event processing systems.

Rather than examining event processing as an individual field, it is important to understand the "big picture." This includes existing, and perhaps even future, communication paradigms, system architectures, and the ability to integrate event processing capabilities into existing systems.

## *Future Work*

The lingering question is "where do we go now?" There really is not any one answer to that, but the event processing community is trying to determine what the next step is. For now, the best answer is to continue research, development, and integration, and work with researchers, developers, and vendors to determine requirements, needs, and possibilities.

There have been two major directions taken so far. The first is to produce off-the-shelf products for event processing needs. However, this will continue to be difficult until more parties with an interest in the technology become involved in the event processing community. The second is to move towards standardization of reference models, terminology, and perhaps even some techniques to end any confusion in the field.

Right now, all stakeholders must focus on understanding and education. The technology must be fully understood. There are a number of design choices that are made when producing an event processing engine and the impact of these decisions on the system must be known. It is also necessary to educate not only those developing the system, but those using and relying on the systems. This will most likely involve creating standard terminology, models, and benchmarks that are understood by both the technical and non-technical communities.

No one can be sure of what the future holds for event processing. However, based on current trends, it is not a fad. Event processing is becoming an essential part of managing and responding to the current situation, and it does not appear that this will change. The only thing left to do is to march forward and see where event processing can be applied and what needs to be done to make it work in that application.

## References


[1] D. Luckham and R. Schulte, “Event Processing Glossary,” [weblog entry] May 2007, [2007 Aug 15], Available at HTTP: http://complexevents.com/?p=195

[2] D. Luckham, et al., “Complex Event Processing,” [weblog] Available at HTTP: http://complexevents.com/

[3] T. Bass, “The Complex Event Processing Blog,” [weblog] Available at HTTP: http://thecepblog.com

[4] [interest group and mailing list] Available at HTTP: http://tech.groups.yahoo.com/group/CEP-Interest/

[5] “The Complex Event Processing Resource Site,” [website] Available at HTTP: http://www.eventstreamprocessing.com/

[6] “Complex Event Processing History,” [webpage] [09 Aug 2007] Available at HTTP: http://www.eventstreamprocessing.com/cep-history.htm

[7] “Stanford Stream Data Manager,” [webpage] Available at HTTP: http://infolab.stanford.edu/stream

[8] “The Aurora Project,” [webpage] Available at HTTP: http://www.cs.brown.edu/research/aurora/

[9] “Borealis Distributed Stream Processing Engine,” [webpage] Available at HTTP: http://www.cs.brown.edu/research/borealis/public/

[10] D. Luckham, The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems, Boston: Addison-Wesley, 2002.

[11] “Coral8, Inc.,” [website] Available at HTTP: http://www.coral8.com

[12] “StreamBase,” [website] Available at HTTP: http://www.streambase.com

[13] “The Telegraph Project at UC Berkeley,” [website] Available at HTTP: http://telegraph.cs.berkeley.edu/index.html

[14] “Welcome to AptSoft Corporation,” [website] Available at HTTP: http://www.aptsoft.com/

[15] “BEA – Business Software, Business Process Management, Service Bus, Service Oriented Architecture,” [website] Available at HTTP: http://www.bea.com/

[16] “EsperTech – Event Stream Intelligence,” [website] Available at HTTP: http://www.espertech.com

[17] “STATStream: a Toolkit for High Speed Statistical Time Series Analysis,” [webpage] Available at HTTP: http://ftp.cs.nyu.edu/shasha/papers/statstream.html

[18] “COUGAR: The Network Is The Database,” [webpage] Available at HTTP: http://www.cs.cornell.edu/database/cougar/index.php

[19] “Cayuga: Stateful Publish/Subscribe for Event Monitoring,” [webpage] Available at HTTP: http://www.cs.cornell.edu/database/cayuga/

[20] L. Gloab and M. Ozsu. “Issues in data stream management.” In SIGMOD Record, Vol. 32, No. 2, pages 5-14, June 2003.

[21] W. Xu, P. Bodik, and D. Patterson. “A Flexible Architecture for Statistical Learning and Data Mining from System Log Stream,” Workshop on Temporal Data Mining: Algorithms, Theory and Applications at the Fourth IEEE International Conference on Data Mining (ICDM) 2004.

[22] A. Widder, R. v. Ammon, P. Schaeffer, and C. Wolff, “Identification of suspicious, unknown event patterns in an event cloud,” ACM International Conference Proceeding

Series, Vol. 233, Proceedings of the 2007 inaugural international conference on Distributed event-based systems. Toronto, Ontario, Canada, 2007, pages 164-170.
[23] T. Bass, "Intrusion or Fraud Detection," [presentation] presented at 2nd Event Processing Symposium. San Mateo, California, 2006.
[24] L. Gurgenm C. Labbe, V. Olive, and C. Roncancio, "A scalable architecture for heterogeneous sensor management," in Proc. 16th International Workshop on Database and Expert Systems Applications, 2005, pages 1108-1112.
[25] D. Baker, D. Georgakopoulos, M. Nodine, A. Chichocki, "Requirements in Providing Awareness from Events."
[26] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik, "Aurora: a new model and architecture for data stream management," The VLDB Journal – The International Journal on Very Large Data Bases, Volume 12, Issue 2, August 2003, pages 120-139.
[27] D. Luckham, "A View of the Current State of Event Processing," [presentation].
[28] J. Hwang, M. Balazinska, A. Rasin, U. Cetintemel, M. Stonebraker, S. Zdonik, "A Comparison of Stream-Oriented High-Availability Algorithms," [technical report CS-03-17], 2003.
[29] P.Th. Eugster, P. Felber, R. Guerraoui, and S.B. Handurukande, "Event systems. How to have your cake and eat it too," In Proc. 22nd International Conference on Distributed Computing Systems Workshops, 2002, pages 625-630.
[30] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, Y. Xing, R. Yan, S. Zdonik, "Aurora: a data stream management system," in Proc. 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, 2003, page 666.
[31] H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, E. Galvez, J. Salz, M. Stonebraker, N. Tatbul, R. Tibbetts, S. Zdonik, "Retrospective on Aurora," The VLDB Journal – The International Journal on Very Large Data Bases, Vol. 13, Issue 4, 2004, pages 370-383.
[32] D. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J. Hwang, W. Linder, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, S. Zdonik, "The Design of the Borealis Stream Processing Engine," in Proceeding of the Second CIDR Conference, January 2005.
[33] S. Zdonik, M. Stonebraker, M. Cherniack, U. Centintemel, M. Balazinska, and H. Balakrishnan, "The Aurora and Medusa Projects," IEEE DE Bulletin, 2003.
[34] J. Nagy, "User-Centric Personalized Extensibility for Data-Driven Web Applications," 2007 IF/AFOSR Minigrant Proposal.
[35] A. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W. White, "Cayuga: A General Purpose Event Monitoring System," In Proceedings of the Third Biennial Conference on Innovative Data Systems Research (CIDR 2007), Asolimar, California, January 2007.
[36] A. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. White, "A General Algebra and Implementation for Monitoring Event Streams," [technical report], July 2005.
[37] K. Vikram, "FingerLakes: A Distributed Event Stream Monitoring System."
[38] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom. "STREAM: The Stanford Data Stream Management System," to

appear in "Data-Stream Management: Processing High-Speed Data Streams," Springer-Verlag, New York, 2005.
[39] "Stream Query Repository," [website] Available at HTTP: http://infolab.stanford.edu/stream/sqr/
[40] S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah, "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World," CIDR, 2003.
[41] S. Krishnamurthy, S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Madden, F. Reiss, M. Shah, "TelegraphCQ: An Architectural Status Report," IEEE Data Engineering Bulletin, Vol. 26, Issue 1, 2003, pages 11-18.
[42] "Coral8 Frequently Asked Questions (Basic)," [product documentation] Available at HTTP/PDF via http://www.coral8.com/developers/documentation.html
[43] "Coral8 Technology Overview," [product documentation] Available at HTTP/PDF via http://www.coral8.com/developers/documentation.html
[44] "Coral8: The Fastest Path to Complex Event Processing," [product documentation] Available at HTTP/PDF via http://www.coral8.com/developers/documentation.html
[45] "Esper performance," [webpage/wiki] Available at HTTP: http://docs.codehaus.org/display/ESPER/Esper+performance
[46] "Apama About Us," [webpage] Available at HTTP: http://www.progress.com/apama/about_us/index.ssp
[47] "Progress Apama – Monitor, Analyze, and Act on Events in Under a Millisecond…," [webpage] Available at HTTP: http://www.progress.com/apama/index.ssp
[48] S. Zdonik, "Stream Processing Overview," [presentation], Workshop on Event Processing, Hawthorne, New York, March 2006.
[49] "Real-Time Data Processing with a Stream Processing Engine," available at HTTP/PDF via http://www.streambase.com/print/knowledgecenter.htm
[50] "Truviso Product Brief," available at HTTP/PDF via http://www.truviso.com/resources/
[51] Y. Zhu and D. Shasha, "StatStream: Statistical monitoring of thousands of data streams in real time," [technical report TR2002-827], New York University, CS Department, New York, New York, 2002.
[52] A. Demers, J. Gehrke, R. Rajaraman, N. Trigoni, and Y. Yao. "The Cougar Project: A Work in Progress Report," On SIGMOD Record, Volume 32, Issue 4, 2003, pages 53-59.
[53] W. Fung, D. Sun, and J. Gehrke, "COUGAR: the network is the database," in Proc. 2002 SIGMOD Conference.
[54] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," ACM SIGMOD Record, Vol. 31, Issue 2, 2002, pages 9-18.
[55] A. Arasu and J. Widom, "A denotational semantics for continuous queries over streams and relations," ACM SIGMOD Record, Volume 33, Issue 4, 2003, pages 6-11.
[56] F. Bry and M. Eckert, "A High-Level Query Language for Events," IEEE Services Computing Workships, 2006, pages 31-38.
[57] A. Arasu, S. Babu, J. Widom, "The CQL continuous query language: semantic foundations and query execution," The VLDB Journal – The International Journal on Very Large Data Bases, Volume 15, Issue 2, 2006, pages 121-142.
[58] "StreamSQL.org," [weblog/website], available at HTTP: http://blogs.streamsql.org
[59] "StreamSQL – Documentation," [webpage], available at HTTP: http://streamsql.org/pages/documentation.html

[60] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. Maskey, et al., “Linear Road: A Stream Data Management Benchmark.”
[61] “Linear Road,” [website], available at HTTP: http://www.cs.brandeis.edu/~linearroad/
[62] P. Tucker, T. Tufte, V. Papadimos, D. Maier. “NEXMark---a Benchmark for Querying Data Streams.” 2002.
[63] “NEXMark Benchmark,” [webpage] Available at HTTP: http://datalab.cs.pdx.edu/niagara/NEXMark/
[64] P. Bizarro, “BiCEP: Benchmarking Complex Event Processing Systems.”
[65] R. Barga, J. Goldstein, M. Ali, and M. Hong, “Consistent streaming through time: A vision for event stream processing,” in proc. Of the 3rd Biennial Conference on Innovative Data Systems Research (CIDR ’07), Asilomar, California, 2007.
[66] P. Eugster, P. Felber, R. Guerraoui, and A. M. Kermarrec, “The many faces of publish./subscribe.”
[67] L. Zeng and H. Lei, “A Semantic Publish/Subscribe System,” in Proc. Of the IEEE International Conference on E-Commerce Technology for Dynamic E-Business, Volume 00, 2004, pages 32